

Real-Time Visualized Freehand 3D Ultrasound Reconstruction Based on GPU

Yakang Dai, Jie Tian, *Fellow, IEEE*, Di Dong, Guorui Yan, and Hairong Zheng, *Member, IEEE*

Abstract—Visualized freehand 3-D ultrasound reconstruction offers to image incremental reconstruction during acquisition and guide users to scan interactively for high-quality volumes. We originally used the graphics processing unit (GPU) to develop a visualized reconstruction algorithm that achieves real-time level. Each newly acquired image was transferred to the memory of the GPU and inserted into the reconstruction volume on the GPU. The partially reconstructed volume was then rendered using GPU-based incremental ray casting. After visualized reconstruction, hole-filling was performed on the GPU to fill remaining empty voxels in the reconstruction volume. We examine the real-time nature of the algorithm using *in vitro* and *in vivo* datasets. The algorithm can image incremental reconstruction at speed of 26–58 frames/s and complete 3-D imaging in the acquisition time for the conventional freehand 3-D ultrasound.

Index Terms—Compute unified device architecture (CUDA), freehand 3-D ultrasound, graphics processing unit (GPU), volume reconstruction, volume rendering.

I. INTRODUCTION

FREEHAND 3-D ultrasound [1] uses a 2-D ultrasound machine and a position sensor to reconstruct a volume. The position sensor (e.g., magnetic sensor [2]–[4], optical sensor [5]) is attached to the probe of the 2-D ultrasound machine to track the position and orientation of the B-scan image, and a set of B-scan images with the relative positions and orientations are acquired and reconstructed to build up the volume. Compared to direct volumetric imaging with a 3-D probe [6], [7], the freehand technique is cheaper and can obtain larger volumes. Therefore, the freehand technique is often used in practical applications including fetal examination [8], [9], neurosurgery [10], [11], etc.

Conventional freehand technique separates the acquisition, reconstruction, and visualization steps [12], which lead to the

following problems: 1) the imaging result is unknown until the acquisition, reconstruction, and visualization are completed (typically 1 min [11], [13]), which disturbs the interactive nature of the ultrasound examination [12] and 2) acquisition feedback is not provided during scanning, consequently, the acquisition quality is highly dependent on scanning experience, which may result in low-quality imaging even rescanning.

One method for addressing the problems is volume reconstruction and visualization during acquisition (visualized reconstruction):

- 1) Perform *incremental volume reconstruction* with the newly acquired image (i.e., insert the image into the volume).
- 2) Perform *incremental volume visualization* (i.e., visualize the partially reconstructed volume), then repeat from step 1.

The method allows us to see which regions of the volume have been reconstructed and where (e.g., have gaps) need further reconstruction while scanning, so that we can scan interactively to obtain a high-quality volume.

Many publications reported visualized reconstruction algorithms. Ohbuchi *et al.* [14], [15] used pixel 3-D kernel interpolation for incremental volume reconstruction and modified ray-casting [16], [17] for incremental volume visualization. However, the algorithm [14], [15] speed was not more than 1 frame/s on a workstation. Edwards *et al.* [18] used a replacement-value pixel distribution method for incremental volume reconstruction and a shear-warp MIP method for incremental volume rendering. Implemented with an imaging board based on the Texas Instruments TMS320C80 multimedia video processor, the visualized reconstruction algorithm [18] could operate at 12.5 frames/s. Rather than performing incremental reconstruction as each B-scan image arrived, Welch *et al.* [19] collected a fixed number of images and then inserted the images into the volume. For incremental volume visualization, they [19] simultaneously displayed cross-sections through the volume and a volume-rendered perspective view. Their algorithm [19] could update the volume and render a new view at 15 frames/s on a Silicon Graphics 320 workstation. Gobbi and Peters [20] implemented a visualized reconstruction algorithm with five parallel threads on a 2 CPU 933 MHz Pentium III workstation. The algorithm [20] could perform incremental reconstruction at maximum 30 frames/s (pixel nearest neighbor), and display three orthogonal slice views through the volume (without volume rendering) at 5 frames/s. Dai *et al.* [21] performed incremental rendering (ray casting) according to the increment of the reconstruction ratio. The visualized reconstruction [21] speed was 12.5 frames/s on a 3.0 GHz Pentium IV PC.

Manuscript received November 24, 2009; revised July 14, 2010; accepted August 22, 2010. Date of publication September 2, 2010; date of current version November 5, 2010. This work was supported in part by NBRPC (2006CB705700, 2011CB707700), in part by CAS HTP, CAS KIP (KSCX2-YW-R-262, KGCX2-YW-129), and in part by NSFC (81042002, 81071218, 30873462, 60910006) in China.

Y. Dai, D. Dong, and G. Yan are with the Medical Image Processing Group, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China.

J. Tian is with the Medical Image Processing Group, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China, and also with the Life Science Center, Xidian University, Xian, Shanxi 710071, China.

H. Zheng is with Paul-C-Lauterbur Research Center for Biomedical Imaging, Institute of Biomedical and Health Engineering, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518067, China.

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TITB.2010.2072993

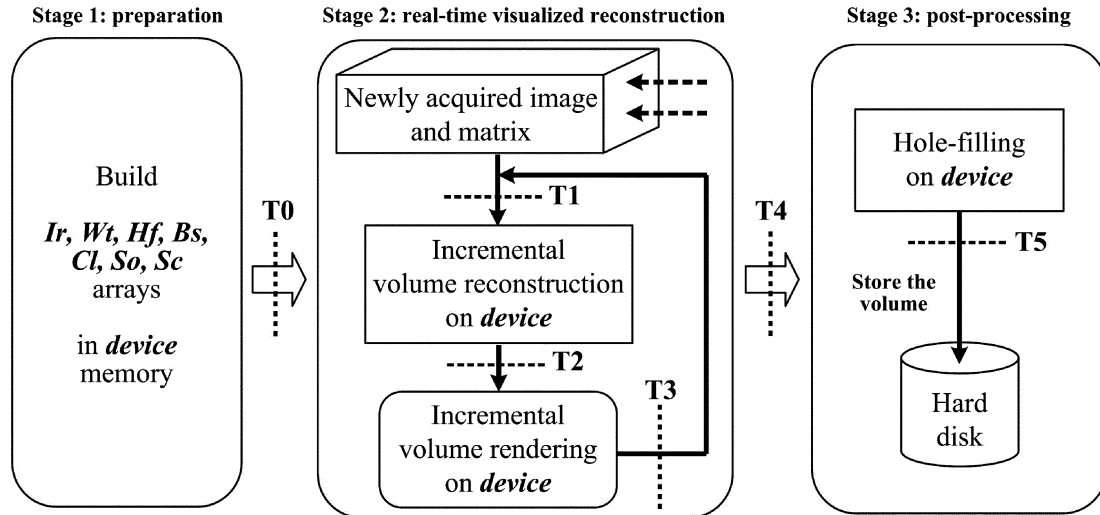


Fig. 1. General flowchart of the real-time visualized reconstruction algorithm. *Ir*, *Wt*, *Hf*, *Bs*, *Cl*, *So* and *Sc* (see Section II-B) were the key arrays prepared in *device* memory for volume reconstruction and rendering. Each image was inserted into the reconstruction volume (incremental volume reconstruction) and the partially reconstructed volume was then rendered (incremental volume rendering). Empty voxels in the reconstruction volume were filled (hole filling) after visualized reconstruction. The incremental volume reconstruction, incremental volume rendering, and hole-filling were all performed on the *device*.

However, the visualized reconstruction is computationally expensive and current algorithms cannot reach real-time level (i.e., the incremental reconstruction and rendering cannot achieve the B-scan image acquisition rate that is typically 25 or 30 frames/s [15], [20], [21]). Nonreal-time visualized reconstruction results in two problems: 1) we cannot get real-time visualized feedback to guide scanning and 2) the 3-D imaging will take long time, which may risk motion (e.g., respiration) artifacts in the reconstructed volume. The problems could highly reduce the speed and quality of the 3-D imaging.

The aim of this paper is to achieve real-time visualized reconstruction and address the aforementioned problems. The paper primarily describes a real-time visualized reconstruction algorithm that implements all expensive computations, including the incremental-volume reconstruction, incremental-volume rendering, and hole filling on the off-the-shelf graphics processing unit (GPU). The algorithm can provide real-time visualized feedback (over 25 frames/s) during acquisition.

II. MATERIALS AND METHODS

A. Hardware and Software

A Windows-PC, with an Intel Core2 1.86 GHz CPU and a NVIDIA GeForce 8800 GT GPU, was used for incremental-volume reconstruction, incremental-volume rendering, etc. The real-time visualized reconstruction algorithm was implemented in C++ and NVIDIA Compute Unified Device Architecture (CUDA) [22]. The incremental-volume rendering module was developed based on the volume rendering framework of a customized Medical Imaging Toolkit [23]. The core of each module in the algorithm was implemented in CUDA and run on the GPU.

For describing the algorithm, we present a brief overview of the execution model of the CUDA. A GPU is regarded as a separate *device* that operates as a coprocessor to the *host*

computer. A C function defined in accordance with the CUDA is called a *kernel*, which, when invoked by a CUDA program on the *host*, is executed on the *device* by a 1-D or 2-D *grid* made up of thread *blocks*. The *blocks* (1-D, 2-D, or 3-D) are distributed to multiprocessors in the *device*. The *threads* within a *block* execute the *kernel* in parallel on one multiprocessor. After all, *blocks* accomplish the execution, the *kernel* is terminated.

B. General Flowchart

Fig. 1 shows the general flowchart of the real-time visualized reconstruction algorithm that consisted of three stages: preparation, real-time visualized reconstruction, and postprocessing. In the first stage, all necessary arrays and variables were prepared for volume reconstruction and rendering. The key arrays were built in the *device* memory, including:

- 1) *Ir*: a 3-D voxel value array for incremental reconstruction and rendering.
- 2) *Wt*: a 3-D voxel weight array for volume reconstruction.
- 3) *Hf*: a 3-D voxel value array for hole filling.
- 4) *Bs*: a 2-D pixel value array for the B-scan image.
- 5) *Cl*: a 2-D color (R, G, B) array for the projection image of the volume.
- 6) *So*: a linear opacity array for the scalar-opacity transfer function.
- 7) *Sc*: a linear color (R, G, B) array for the scalar-color transfer function.

The *Hf*, *Bs*, *So*, and *Sc* were bound to textures, the *Ir* was bound to a texture when it was used for volume rendering, and the *Wt* was bound to a texture when it was used for hole filling. In the second stage, for each newly acquired image and its transformation matrix from the position sensor, the incremental volume reconstruction and rendering were performed on the *device*. After the visualized reconstruction, there may be remaining unfilled voxels in the volume. Therefore, in the third stage,

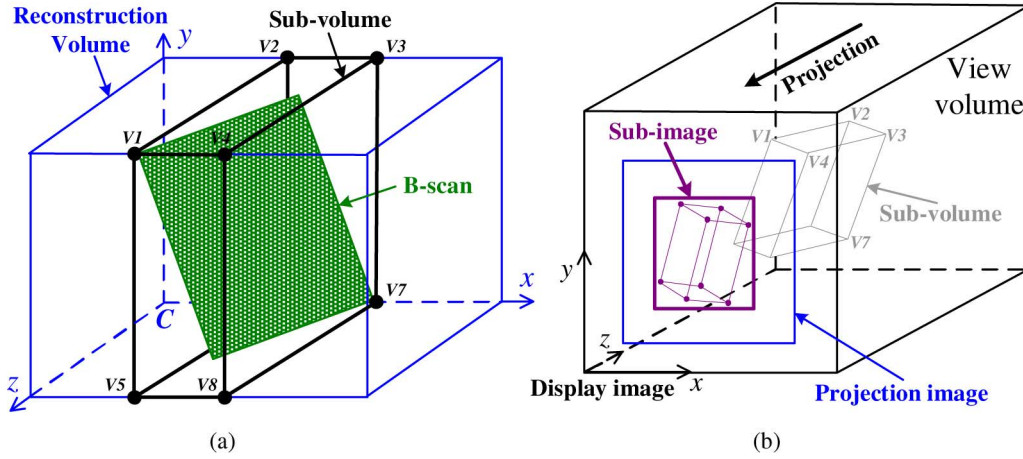


Fig. 2. Subvolume and subimage computations. (a) Coordinates of the four vertices of the B-scan image in the reconstruction volume coordinate system C were computed to determine a compact sub-volume. (b) Projection coordinates of the eight vertices of the sub-volume in the display image were computed to determine a compact subimage.

an additional hole-filling operation was performed to fill empty voxels. Finally, the reconstructed volume could be transferred to the *host* computer and stored on the hard disk. In the following, we introduce the implementation details of the incremental-volume reconstruction, incremental-volume rendering, and hole filling.

C. Incremental Volume Reconstruction

First, the transformation matrix from the B-scan coordinate system to the reconstruction volume coordinate system ${}^C T_P$ (used for transforming the B-scan image to the reconstruction volume) was figured out by [24]

$${}^C T_P = {}^C T_T \cdot {}^T T_R \cdot {}^R T_P \quad (1)$$

on the *host*, where P is the coordinate system attached to each *B-scan image*, R and T (subscript or superscript) are the coordinate systems of the *receiver* and *transmitter* of the position sensor, respectively, C is the coordinate system of the *reconstruction volume*, ${}^R T_P$, ${}^T T_R$, ${}^C T_T$, and ${}^C T_P$ can be written as a uniform format ${}^J T_I$, which denotes the transformation matrix from the coordinate system I to the coordinate system J . The ${}^R T_P$ and ${}^C T_T$ can be predetermined by spatial calibration [25] and semiautomatic definition [26] respectively. Second, the 2-D image and ${}^C T_P$ were transferred to the *device* memory (the image was stored in the 2-D pixel value array Bs , see Section II-B). Finally, an incremental reconstruction *kernel* was called to insert the image into the volume.

The incremental reconstruction *kernel* was executed on the *device* by a 2-D incremental reconstruction *grid* to insert the image in parallel. Neighboring pixels in the image may contribute to the same voxel in the reconstruction volume. If each *thread* processes a pixel, multiple *threads* may access the same voxel in parallel, which may cause parallel insertion errors. To avoid the errors, each *thread* in the *block* processed 4×4 pixels (the size was selected by experiment) in order. Assume the size of each *block* in the *grid* was $M \times N$ (we used 16×16 to achieve fast speed), and the size of the image was $W \times H$,

then the number of *blocks* in the *grid* was $(W/4M) \times (H/4N)$, and the image could be inserted parallelly by $(W/4) \times (H/4)$ *threads*. For each pixel (i, j) of the B-scan image with the value $Bs(i, j)$, first, the pixel location in the reconstruction volume was calculated by

$${}^C X = {}^C T_P \cdot {}^P X \quad (2)$$

where ${}^P X$ and ${}^C X$ are the coordinates of the pixel in P (the B-scan coordinate system) and C (the reconstruction volume coordinate system), respectively. Then, the pixel was distributed into the volume using pixel 3-D kernel interpolation [1]. The 3-D kernel we used was a $2 \times 2 \times 2$ cube [18], [20] (the size was used as a tradeoff between interpolation quality and speed) and the pixel contributed to eight neighboring voxels. For each neighboring voxel (m, n, l) , the value $Ir(m, n, l)$ and weight $Wt(m, n, l)$ were updated by

$$\begin{cases} \text{sum} = Ir(m, n, l) \cdot Wt(m, n, l) + Bs(i, j) \cdot \text{inv}D \\ Wt(m, n, l) = Wt(m, n, l) + \text{inv}D \\ Ir(m, n, l) = \text{sum}/Wt(m, n, l) \end{cases} \quad (3)$$

where $\text{inv}D$ is the inverse distance between the voxel (m, n, l) and the pixel (i, j) .

D. Incremental Volume Rendering

We used ray casting to render the volume. The ray casting referred to four coordinate systems, including the *reconstruction volume* coordinate system C [see Fig. 2(a)], *world* coordinate system, *view* (also camera) coordinate system, and *display image* coordinate system D [see Fig. 2(b)]. The reconstruction volume and camera were set in the world coordinate system. The camera faced the reconstruction volume, and used orthogonal (also parallel) projection to set a view volume [see Fig. 2(b)] to surround the reconstruction volume. The display image, whose size was the same as the display window size, corresponded to the near plane of the view volume.

After inserting the newly acquired B-scan image into the reconstruction volume, some voxels in the reconstruction volume were updated. Projecting the subvolume enclosing the updated voxels onto the display image, we could get a subimage in the projection image of the volume [see Fig. 2(b)]. Actually, only the subimage need to be updated, therefore, we merely performed ray casting for the pixels in the subimage, and kept other pixels in the projection image unchanged. The method can highly accelerate the incremental volume rendering if the subimage is small. Following is the implementation flow of the incremental volume rendering. First, the position and size of the subimage in the projection image were figured out on the *host*. Second, the position and size were transferred to the *device* memory. Finally, a ray-casting *kernel* was called to update the subimage.

The position and size of the subimage were figured out by two steps: subvolume computation and subimage computation (see Fig. 2). In the subvolume computation, first, we computed the coordinates of the four vertices of the newly acquired image in the reconstruction volume coordinate system by (2). Second, with the minimal and maximal x , y , and z elements of the four coordinates, we determined a compact cube (eight vertices) enclosing the B-scan image. Finally, we performed clamping operations to ensure the subvolume was in the reconstruction volume. In the sub-image computation, first, we computed the projection coordinates (x and y elements of the ${}^D X$) of the eight vertices of the subvolume in the display image by

$${}^D X = {}^D T_C \cdot {}^C X \quad (4)$$

where ${}^D T_C$ is the transformation matrix from the reconstruction volume coordinate system C to the display image coordinate system D , ${}^C X$ and ${}^D X$ are the coordinates of the vertex in the C and D , respectively. Second, with the minimal and maximal x and y elements of the eight coordinates in D , we determined a compact rectangle (four vertices) enclosing the pixels required to be updated. Finally, we performed clamping operations to ensure the subimage was in the projection image (the position and size of the projection image of the reconstruction volume in the display image were precomputed in the preparation stage).

The incremental ray-casting *kernel* was executed on the *device* by a 2-D incremental ray-casting *grid* to update the subimage in parallel. Each pixel in the subimage was processed by a *thread* (the *block* sizes for the translation sequence and fan sequence are 2×32 and 16×16 , respectively, see Section III. The sizes were selected by experiment to achieve fast speed). Each *thread* cast a ray from the associated pixel (i, j) along the z axis of the display image coordinate system through the reconstruction volume and performed front-to-back sampling and recursive compositing [27] to compute the color of the pixel. Assume S and E are the start and end intersections between the ray and the reconstruction volume, ${}^C X_S$ denotes the coordinates of S in the reconstruction volume coordinate system C , r is the normal of the ray in C , and d_{SE} is the distance between the S and E in C . Then, the computation details of the pixel color $Cl(i, j)$ are as follows:

```

d = 1;
 ${}^C X_{Sample} = {}^C X_S + r$ ;
while  $d < d_{SE}$  and  $\alpha < 0.98$  do
  Find the nearest voxel  $(m, n, l)$  around  ${}^C X_{Sample}$ ;
   $v = Ir(m, n, l)$ ;
  if  $So(v) > 0$  then
     $c = Sc(v) \cdot So(v) \cdot (1 - \alpha) + c$ ;
     $\alpha = So(v) \cdot (1 - \alpha) + \alpha$ ;
  end
   $d = d + 1$ ;
   ${}^C X_{Sample} = {}^C X_{Sample} + r$ ;
end
if  $\alpha > 0$  then
   $Cl(i, j) = c/\alpha$ ;
end

```

where ${}^C X_{Sample}$ denotes the coordinates of the sample in the reconstruction volume coordinate system C , $Ir(m, n, l)$ is the voxel value, $Sc(v)$ and $So(v)$ are the color and opacity of the voxel value v , respectively (see Section II-B for the definitions of Ir , Sc , So , and Cl), c and α are the accumulated color and accumulated opacity, respectively.

E. Hole filling

After visualized reconstruction, the values of the 3-D voxel value array (Ir) for incremental reconstruction and rendering were copied into the 3-D voxel value array (Hf) for hole filling. In the hole filling, all slices of the reconstructed volume were traversed in order. For each slice, a hole-filling *kernel* was called to fill empty voxels in the slice. The hole-filling *kernel* was executed on the *device* by a 2-D hole-filling *grid*. Each voxel in the slice was processed by a *thread* (the *block* size we used is 16×16). Each *thread* detected the associated voxel (m, n, l) , and filled the voxel with the average of the neighboring nonzero voxels, if the voxel was empty ($Wt(m, n, l) == 0$):

```

if  $Wt(m, n, l) == 0$  then
   $sum = 0$ ;
   $number = 0$ ;
  foreach voxel  $(i, j, k)$  in the neighborhood do
    if  $Hf(i, j, k) > 0$  then
       $sum = sum + Hf(i, j, k)$ ;
       $number = number + 1$ ;
    end
  end
  if  $number > 0$  then
     $Ir(m, n, l) = sum/number$ ;
  end
end

```

where Wt is the 3-D voxel weight array for volume reconstruction [see Section II-B and (3)] and the neighborhood size we used is $3 \times 3 \times 3$ (B-scan images were densely acquired and 3-D kernel interpolation was used for incremental reconstruction, therefore, few empty voxels were remaining between

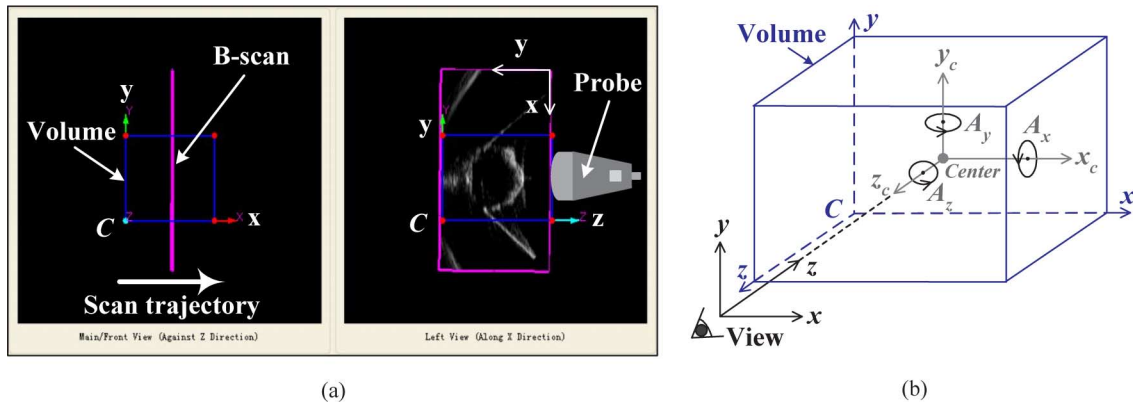


Fig. 3. Setting of reconstruction volume coordinate system C and view coordinate system V for translation sequence. $x_c y_c z_c$ is the center coordinate system of the reconstruction volume [see (b)], the origin of V is on the z_c axis, the z axis of V is opposite to the z_c axis, and the x axis of V is parallel to the x_c axis. The setting achieves an appropriate view direction for interactive scanning [see (a)] and small subimage sizes for fast incremental rendering.

TABLE I
TIME SPENT IN MAIN STEPS OF THE ALGORITHM

Time (ms)	Preparation	Incremental Reconstruction	Incremental Rendering	Hole-filling
Translation sequence	21	11	6	15
Fan sequence	22	21	5	46

B-scan images after visualized reconstruction and the neighborhood size was cost effective for the hole filling).

III. RESULTS

Two datasets are used to evaluate the incremental volume reconstruction, incremental volume rendering and hole filling. One dataset is 1000 B-scan images from an *in vitro* phantom (plastic elephant) [26] by translation scan, and the other is 135 B-scan images from an *in vivo* human liver [28] by fan scan. The images are visualizedly reconstructed to volumes with the described algorithm. As shown in Fig. 1, the time spent in the preparation (T_0), incremental volume reconstruction (T_2-T_1), incremental volume rendering (T_3-T_2), and hole filling (T_5-T_4) are recorded.

A. Evaluation With Translation Sequence

The *in vitro* dataset is collected using image guided acquisition, and the B-scans are approximately parallel to the zoy plane in the reconstruction volume coordinate system C [see Fig. 3(a)]. The B-scan image size and volume size are 552×274 and $256 \times 256 \times 256$, respectively, and the display image size is 512×512 . As shown in Fig. 3(b), $x_c y_c z_c$ is the center coordinate system of the reconstruction volume, the origin of the view (also camera) coordinate system V (see Section II-D) is on the z_c axis, the z axis of V is opposite to the z_c axis, and the x axis of V is parallel to the x_c axis. The aforesaid setting enables us to see gaps between translation B-scans, which can well guide translation scan, so we name the setting as translation scan reference setting.

Table I shows the time spent in each main step of the algorithm under the reference setting. The speed of the preparation, incremental reconstruction (90 frames/s) and hole filling are primarily dependent on the B-scan image size and volume size (in

inverse proportion), while the speed of the incremental rendering mainly lies on the volume size and subimage size (in inverse proportion as well). Under the reference setting, the subimage size (average 13×215) is small and a few *threads* are needed (the GPU we used can run 112 *threads* concurrently), which enables the incremental rendering to reach 166 frames/s. The incremental volume reconstruction and rendering speed (58 frames/s) is much faster than the real-time level (25 frames/s). Fig. 4 shows the incremental volume rendering results after 100, 200, 300, 400, and 500 images are inserted into the volume.

On the basis of the reference setting, we have the volume only rotate about the x_c (y_c or z_c) axis by an angle A_x (A_y or A_z) [see Fig. 3(b)], and perform visualized reconstruction under the new setting. Fig. 5(a) illustrates speed of the incremental rendering at different A_x (A_y and A_z are 0), A_y (A_x and A_z are 0), or A_z (A_x and A_y are 0). As the incremental volume reconstruction takes 11 ms (see Table I), to achieve real-time visualized reconstruction, the incremental volume rendering must be completed in 29 ms. For different A_x or A_z , the subimage size is small, and the incremental volume rendering can be accomplished in 6–11 ms [see Fig. 5(a)], making the incremental reconstruction and rendering achieve 45–58 frames/s. For different A_y , the incremental volume rendering can finish in 7–23 ms, which makes the incremental reconstruction and rendering reach 29–55 frames/s.

B. Evaluation With Fan Sequence

The *in vivo* dataset is from the Medical Imaging Group, University of Cambridge [28]. As shown in Fig. 6, the x axis of the reconstruction volume coordinate system C corresponds to the scan trajectory of the B-scans, the y axis of C faces the overall y (longitudinal) axis of the B-scans, and the z axis of C faces the overall x (lateral) axis of the B-scans. The B-scan image

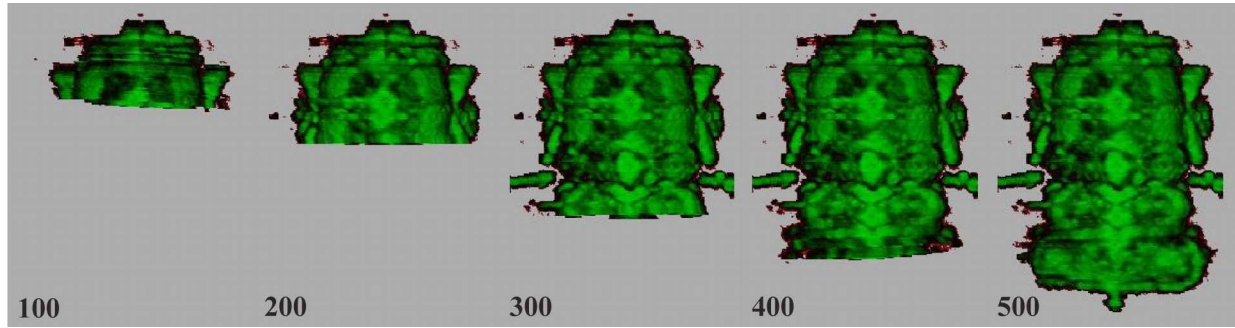


Fig. 4. Visualized reconstruction for translation sequence. The incremental reconstruction is imaged at 58 frames/s. A set of incremental rendering results (after 100, 200, 300, 400, and 500 images are inserted into the volume) is shown.

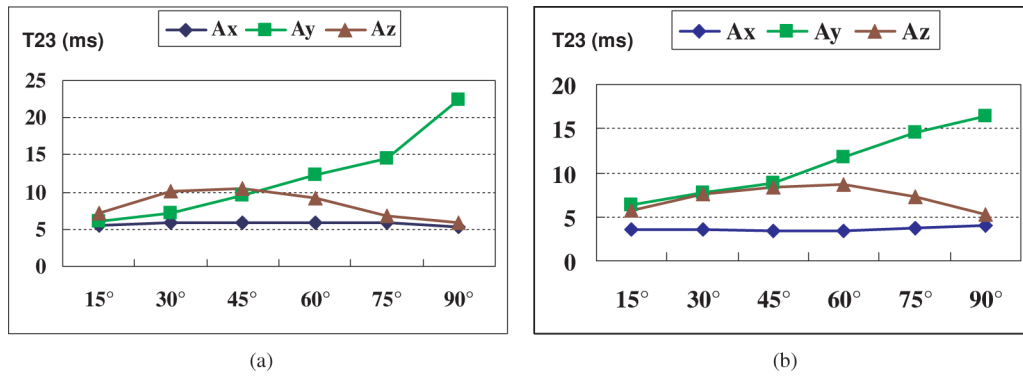


Fig. 5. Incremental volume rendering speed at different rotation angles for (a) translation sequence and (b) fan sequence. The subimage size for different A_x is small, therefore, the incremental rendering speed is very fast and stable. The subimage size grows when A_y increases, so the incremental rendering speed rises until A_y reaches around 90° . The subimage size grows until A_z reaches around 45° and decreases until A_z reaches around 90° , thus, the incremental rendering speed first rises and then gradually tapers off. Speed differences between the translation sequence and fan sequence are caused by volume contents, *block* size, and *thread* scheduling managed by CUDA.

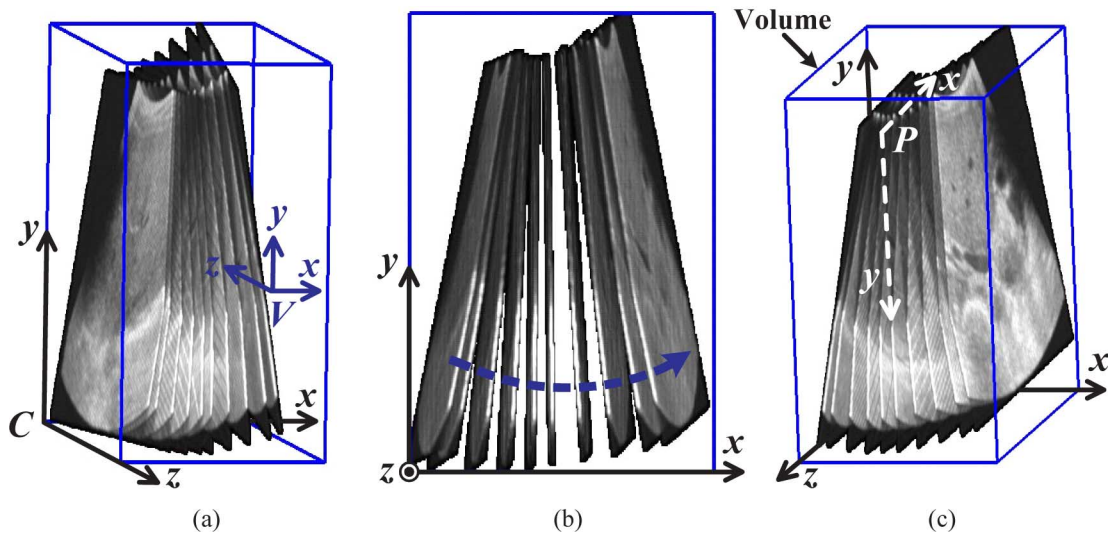


Fig. 6. Setting of the reconstruction volume coordinate system C and view coordinate system V for fan sequence. The origin of V is on the z_c axis of the center coordinate system $[x_c y_c z_c]$, see Fig. 3(b)] of the reconstruction volume, the z axis of V is opposite to the z axis of C , and the x axis of V is parallel to the x axis of C . The setting achieves an appropriate view direction for interactive scanning and small subimage sizes for fast incremental rendering.

size and volume size are 480×413 and $256 \times 256 \times 256$, respectively, and the display image size is 512×512 . Similarly, we set the origin of the view coordinate system V on the z_c axis, the z axis of V opposite to the z_c axis, and the x axis of V parallel to the x_c axis [see Figs. 3(b) and 6(a)]. The aforesaid

setting enables us to see gaps between fan B-scans, so similarly we call the setting as fan scan reference setting.

The time spent in each main step of the algorithm under the reference setting is shown in Table I. The subimage size is average 34×238 , enabling the incremental rendering to reach

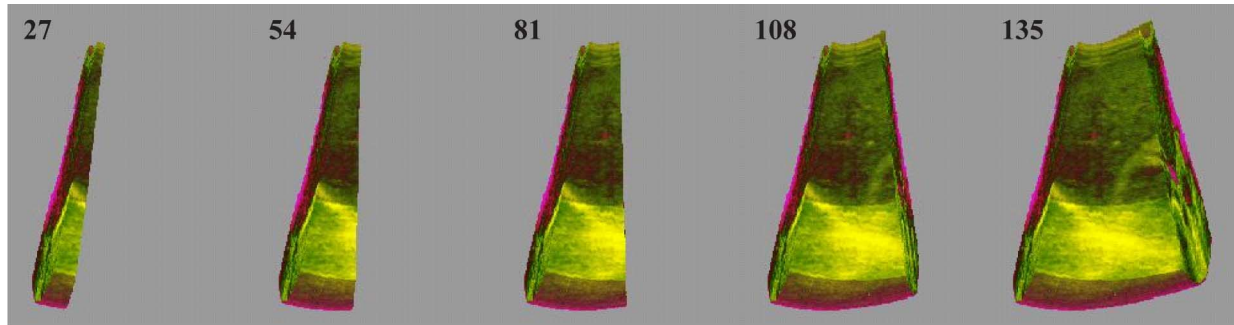


Fig. 7. Visualized reconstruction for fan sequence. The incremental reconstruction is imaged at 38 frames/s. A set of incremental rendering results (after 27, 54, 81, 108, and 135 images are inserted into the volume) is shown.

200 frames/s. The incremental volume reconstruction and rendering can reach 38 frames/s, which is faster than the real-time level (25 frames/s). Fig. 7 shows the incremental volume rendering results after 27, 54, 81, 108, and 135 images are inserted into the volume.

Similarly, based on the reference setting, we have the volume only rotate about the x_c (y_c or z_c) axis by an angle A_x (A_y or A_z), and perform visualized reconstruction under the new setting. Speed of the incremental rendering at different A_x , A_y , or A_z is illustrated in Fig. 5(b). For different A_x or A_z , similarly the subimage size is small, and the incremental rendering can be completed in 4–9 ms [see Fig. 5(b)], making the incremental reconstruction (takes 21 ms, see Table I) and rendering achieve 33–40 frames/s. For different A_y , the incremental rendering can finish in 7–17 ms, which makes the incremental reconstruction and rendering reach 26–35 frames/s.

IV. CONCLUSION AND FUTURE PERSPECTIVE

We have presented the implementation of a real-time visualized reconstruction algorithm and evaluated the algorithm with an *in vitro* dataset based on translation scan and an *in vivo* dataset based on fan scan. The algorithm uses the powerful but cheap GPU to implement the time-consuming reconstruction and rendering computations, which highly accelerates the visualized reconstruction. The evaluations demonstrate the incremental volume reconstruction and rendering speed can exceed the B-scan image acquisition rate (25 frames/s). Using the algorithm, we can not only get real-time visualized feedback on the acquisition and reconstruction during scanning, but also complete the 3-D imaging within the data acquisition time for the conventional freehand 3-D ultrasound.

For high-quality imaging, we need see gaps between B-scans during scanning, so as to well guide the scan and acquisition. Therefore, the x axis of the view coordinate system V should correspond to the scan trajectory, and the view direction (z axis of V) should correspond to the overall x or y axis of the B-scans. The setting makes the subimages of the B-scans in the display image small, which can greatly speed up the incremental volume rendering. Our evaluations show the incremental volume reconstruction and rendering can exceed 30 frames/s by using the setting. In addition to translation scan and fan scan, the setting benefits rotation scan as well. Actually, the evaluations

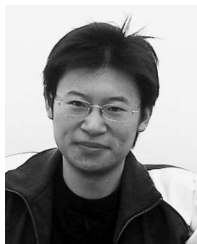
at different A_z for the translation and fan sequences resemble the evaluation for a rotation sequence under the setting.

The technically developed but promising real-time visualized reconstruction algorithm is potentially useful for practical applications, especially freehand 3-D ultrasound guided surgery, where the speed and quality of the 3-D imaging are critical. We are developing a real-time freehand system based on the algorithm and will try to apply the system to clinical applications in the future.

REFERENCES

- [1] O. V. Solberg, F. Lindseth, H. Torp, R. E. Blake, and T. A. N. Hernes, "Freehand 3D ultrasound reconstruction algorithms—A review," *Ultrasound Med. Biol.*, vol. 33, no. 7, pp. 991–1009, 2007.
- [2] D. F. Leotta, P. R. Detmer, and R. W. Martin, "Performance of a miniature magnetic position sensor for three-dimensional ultrasound imaging," *Ultrasound Med. Biol.*, vol. 23, no. 4, pp. 597–609, 1997.
- [3] S. Berg, H. Torp, D. Martens, E. Steen, S. Samstad, I. Høivik, and B. Olstad, "Dynamic three-dimensional freehand echocardiography using raw digital ultrasound data," *Ultrasound Med. Biol.*, vol. 25, no. 5, pp. 745–753, 1999.
- [4] S. Meairs, J. Beyer, and M. Hennerici, "Reconstruction and visualization of irregularly sampled three- and four-dimensional ultrasound data for cerebrovascular applications," *Ultrasound Med. Biol.*, vol. 26, no. 2, pp. 263–272, 2000.
- [5] J. W. Trobaugh, D. J. Trobaugh, and W. D. Richard, "Three-dimensional imaging with stereotactic ultrasonography," *Comput. Med. Imaging Graph.*, vol. 18, pp. 315–323, 1994.
- [6] O. T. von Ramm, S. W. Smith, and H. G. Pavy, Jr., "High-speed ultrasound volumetric imaging system. Part II: Parallel processing and image display," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 38, no. 2, pp. 109–115, Mar. 1991.
- [7] E. Merz, F. Bahlmann, G. Weber, and D. Macchiella, "Three-dimensional ultrasonography in prenatal diagnosis," *J. Perinatal Med.*, vol. 23, no. 3, pp. 213–222, 1995.
- [8] E. Esh-Broder, F. B. Ushakov, T. Imbar, and S. Yagel, "Application of freehand three-dimensional echocardiography in the evaluation of fetal cardiac ejection fraction: A preliminary study," *Ultrasound Obstet. Gynecol.*, vol. 23, no. 6, pp. 546–551, 2004.
- [9] U. Herberg, H. Goldberg, and J. Breuer, "Three- and four-dimensional freehand fetal echocardiography: A feasibility study using a hand-held Doppler probe for cardiac gating," *Ultrasound Obstet. Gynecol.*, vol. 25, no. 4, pp. 362–371, 2005.
- [10] G. Unsgaard, S. Ommedal, T. Muller, A. Gronningsaeter, and T. A. N. Hernes, "Neuronavigation by intraoperative three-dimensional ultrasound: Initial experience during brain tumor resection," *Neurosurgery*, vol. 50, pp. 804–812, 2002.
- [11] G. Unsgaard, O. M. Rygh, T. Selbekk, T. B. Muller, F. Kolstad, F. Lindseth, and T. A. N. Hernes, "Intra-operative 3D ultrasound in neurosurgery," *Acta Neurochir.*, vol. 148, pp. 235–253, 2006.
- [12] R. W. Prager, A. Gee, and L. Berman, "Stradx: Real-time acquisition and visualisation of freehand 3D ultrasound," *Med. Image Anal.*, vol. 3, no. 2, pp. 129–140, 1998.

- [13] O. M. Rygh, T. A. Nagelhus Hernes, F. Lindseth, T. Selbekk, T. Brostrup Müller, and G. Unsgaard, "Intraoperative navigated 3-dimensional ultrasound angiography in tumor surgery," *Surgical Neurology*, vol. 66, no. 6, pp. 581–592, 2006.
- [14] R. Ohbuchi and H. Fuchs, "Incremental volume rendering algorithm for interactive 3D ultrasound imaging," in *Proc. 12th Int. Conf. Inf. Process. Med. Imaging*, 1991, pp. 486–500.
- [15] R. Ohbuchi, D. Chen, and H. Fuchs, "Incremental volume reconstruction and rendering for 3D ultrasound imaging," in *SPIE Proc. Vis. Biomed. Comput.*, vol. 1808, 1992, pp. 312–323.
- [16] M. Levoy, "Display of surfaces from volume data," *IEEE Trans. Comput. Graph. Appl.*, vol. 8, no. 3, pp. 29–37, May 1988.
- [17] R. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," *Comput. Graph.*, vol. 22, no. 4, pp. 65–74, 1988.
- [18] W. S. Edwards, C. Deforge, and Y. Kim, "Interactive three-dimensional ultrasound using a programmable multimedia processor," *Int. J. Imaging Syst. Technol.*, vol. 9, no. 6, pp. 442–454, 1998.
- [19] J. N. Welch, J. A. Johnson, M. R. Bax, R. Badr, and R. Shahidi, "A real-time freehand 3-D ultrasound system for image-guided surgery," in *Proc. IEEE Ultrasound Symp.*, San Juan, Puerto Rico, vol. 2, 2000, pp. 1061–1064.
- [20] D. G. Gobbi and T. M. Peters, "Interactive intra-operative 3d ultrasound reconstruction and visualization," in *Proc. Med. Image Comput. Comput. Assisted Intervention (MICCAI)*, Tokyo, Japan, Springer, vol. 2489, 2002, pp. 156–163.
- [21] Y. Dai, J. Tian, J. Xue, and J. Liu, "A qualitative and quantitative interaction technique for freehand 3D ultrasound imaging," in *Proc. IEEE Eng. Med. Biol. Soc. (EMBS)*, New York, 2006, pp. 2750–2753.
- [22] C. NVIDIA, *NVIDIA compute unified device archit. programming guide version 2.0*, NVIDIA Corporation, Santa Clara, CA, 2008. [Online]. Available: http://www.nvidia.com/object/cuda_develop.html
- [23] J. Tian, J. Xue, Y. Dai, J. Chen, and J. Zheng, "A novel software platform for medical image processing and analyzing," *IEEE Trans. on Inf. Technol. Biomed.*, vol. 12, no. 6, pp. 800–812, Nov. 2008.
- [24] R. Rohling, A. Gee, and L. Berman, "Three-dimensional spatial compounding of ultrasound images," *Med. Image Anal.*, vol. 1, no. 3, pp. 177–193, 1997.
- [25] L. Mercier, T. Langøf, F. Lindseth, and L. D. Collins, "A review of calibration techniques for freehand 3-d ultrasound systems," *Ultrasound Med. Biol.*, vol. 31, no. 4, 2005.
- [26] Y. Dai, J. Tian, and J. Zheng, "Semiautomatic determination of the reconstruction volume for real-time freehand 3d ultrasound reconstruction," in *SPIE Med. Imaging*, February 8, 2009, Lake Buena Vista, Florida, vol. 7265, 2009, pp. 72651E-1–72651E-8.
- [27] A. Kaufman and K. Mueller, *Overview of Volume Rendering*. Chapter for the Visualization Handbook, C. Johnson and C. Hansen, Eds., Academic Press, 2005.
- [28] Hepatic. Medical Imaging Group, University of Cambridge. [Online]. Available: <http://mi.eng.cam.ac.uk/~rwp/stradwin/>



Yakang Dai received the Ph.D. degree in computer applied technology from Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2009.

He is a member in the Medical Image Processing Group, Institute of Automation, Chinese Academy of Sciences. He was involved in the development of Medical Imaging Toolkit (MITK) and 3-D Medical Image Processing and Analyzing System (3DMed). His current research interests include 3-D ultrasound imaging, visualization and software platform for

medical image computing.

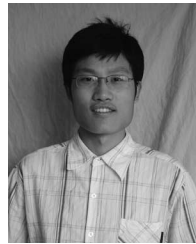


Jie Tian received the Ph.D. degree (Hons.) in artificial intelligence from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 1992.

During 1995–1996, he was a Postdoctoral Fellow at the medical image processing group, University of Pennsylvania. Since 1997, he has been a Professor in the Medical Image Processing group of the Institute of Automation, Chinese Academy of Sciences. His current research interests include the medical image process and analysis, pattern recognition. He is the

author or coauthor of more than 80 research papers in the international journals and conferences.

Dr. Tian is the Beijing Chapter Chair of The Engineering in Medicine and Biology Society of the IEEE.



Di Dong received the B.S. degree in automation from the University of Science and Technology Beijing, Beijing, China, in 2008. He is currently working toward the Ph.D. degree in the Medical Image Processing Group, Institute of Automation, Chinese Academy of Sciences, Beijing, China.

Since 2008, he has been involved in the development of Medical Imaging Toolkit (MITK) and 3-D Medical Image Processing and Analyzing System (3DMed). His current research interests include freehand ultrasound imaging and optical projection

tomography.



Guorui Yan received the B.S. degree in information engineering from Xi'an Jiaotong University, Xi'an, China, in 2006.

He is currently a member in the Medical Image Processing Group, Institute of Automation, Chinese Academy of Sciences, Beijing, China. Since 2006, he has been involved in the development of Medical Imaging Toolkit (MITK) and 3-D Medical Image Processing and Analyzing System (3DMed). His current research interest is CT image reconstruction.



Hairong Zheng received the Ph.D. degree in mechanical engineering, Biomedical Engineering Division from University of Colorado at Boulder, Boulder, in 2006.

He was with the University of California, Davis as a Postdoctoral Fellow and Project Scientist in the Biomedical Engineering Department. He is currently a Professor and Executive Director of the Paul C. Lauterbur Research Center for Biomedical Imaging, Deputy Director of Institute of Biomedical and Health Engineering, Shenzhen Institutes of Advanced Tech-

nology, Chinese Academy of Sciences. His research interests include developing multimodality imaging and advanced biomedical ultrasound system for imaging, drug delivery, and therapy. He is the author or coauthor of more than 70 peer-reviewed journal papers and international conference proceedings.